Hello World

Week 2

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❏ What it means to write a Hello World;
- ❏ Use the Command Window for basic code execution;
- ❏ Write a simple script in the Editor.
- ❏ Format variables in output messages;
- ❏ The various formatting codes in MATLAB.

Table of Contents (ToC)

1 – What is “Hello World”

In programming, the “Hello World” script is considered a *minimum working example*.

It is a simple bit of code to demonstrate that you can:

- Create and edit a script;
- Successfully execute the script;
- Successfully print to the screen.

Printing content is highly valuable in code development and debugging.

2 – Using the `disp` Command

The Command Window in MATLAB is a good place to test one-line bits of code.

To print the message “Hello World” to screen, type the following and then hit “Enter” on your keyboard:

```
>> disp('Hello World')
```

Command Window

```
Hello World
```

Command Window Output

2 – Multiple Ways to Print

As it turns out, there are multiple MATLAB commands that can print text.

```
disp('Hello World')     % Display value of variable
fprintf('Hello World') % Write data to text file (or Command Window)
```

`fprintf` is recommended for formatting output strings.

As this lecture will demonstrate, **`fprintf`** enables printing more complex messages than **`disp`**.

2 – `disp` and `fprintf`

With such a simple string input, both **`disp`** and **`fprintf`** display identical messages.

```
>> disp('Hello World')
Hello World

>> fprintf('Hello World')
Hello World
```

Command Window

This will change in future slides.

3 – Creating a Script

To translate this into a script, first recall how to create a script from Lecture 1.2.

Create and save a new script with the name **`hello_world.m`** for us to continue working with this Hello World example.

To build on the Command Window executions, the complexity will be increased slightly in this script.

3 – Using a Variable

By passing a variable as the input, we can avoid typing the same string multiple times.

```
core_message = 'hello world - from MEMS 1140';
disp(core_message)
fprintf(core_message)
```

MATLAB Script (.m)

Here, the variable **`core_message`** stores the text. It is then referenced in both commands as an input.

3 – Script Output

Running this script produces the following output:

```
>> hello_world.m
hello world - from MEMS 1140
hello world - from MEMS 1140
```

Command Window Output

Again, with such a simple string, both **`disp`** and **`fprintf`** produce identical results.

4 – Log Messages in a Script

In addition to printing simple strings, **`fprintf`** enables formatting data from variables into the message.

This is valuable in printing log messages to track the values of key variables throughout a script.

For example, in a bridge optimization script, it can be helpful to print performance characteristics throughout.

4 – Example Log Messages

The following snippet is taken from such a log:

```
At angle = 75, bridge held load of 4870 [N]
Maximum Performance Index of 1.623 at an angle 42 degrees
Bridge expected to support 3380.00 [N] at this optimal angle
```

Command Window Output

First, it logs the load capability of the final bridge configuration.

Then, it logs the maximum performance index , its associated angle, and its expected load.

4 – Formatting Codes

The **`fprintf`** function accepts many formatting codes that specify things like data types, degree of precision, number of total characters, and more.

The following are just a few that are likely to be the most relevant for this course:

```
fprintf('The number %d', 4)          % %d encodes integer values
fprintf('The number %f', 4.0)        % %f encodes float values
fprintf('The message %s', 'hi')      % %s encodes string values
```

4 – Formatting First Message

Let's return to the example log message from earlier.

The first line contains two numeric values:

```
At angle = 75, bridge held load of 4870 [N]
```

The angle and the load are both printed by extracting the values of integer variables in the script.

```
fprintf('At angle = %d, bridge held load of %d [N]', theta, load)
```

4 – Formatting Second Message

The second line contains two numeric values:

```
Maximum Performance Index of 1.623 at an angle 42 degrees
```

The maximum performance index value and the angle at which it was found are both extracted from variables in the script.

```
fprintf('Maximum Performance Index of %.3f, at an angle %d degrees', ...
    max_pi, optimal_angle)
```

Adding ellipses (. . .) into your code enables a line break, which can make code more readable.

4 – Formatting Third Message

The third line contains just one numeric value:

```
Bridge expected to support 3380.00 [N] at this optimal angle
```

The predicted supported load is extracted from a variable in the script.

```
fprintf('Bridge expected to support %.2f [N] at this optimal angle', ...
    load_at_max_pi_angle)
```

5 – Formatting Codes

For an explicit review of the formatting codes, Mathworks provides the following information:

To format a variable, it ***must*** include the leading % character, as well as a trailing conversion character.

Examples earlier utilized `%d`, `%f`, and `%s`, but there are more.

The other fields are filled in the following order:

```
<Identifier><Flags><Field Width><Precision><Subtype>
```

5 – String Formatting — Identifier

The numeric identifier instructs the formatted output to print a specific argument of the **`fprintf`** command.

For example, consider the following two print statements:

```
>> fprintf('%1$f', 15, 9.876)
15.000000
>> fprintf('%2$f', 15, 9.876)
9.876000
```

Command Window

The identifier specifies whether to print argument 1 or 2.

5 – String Formatting — Flags

There are five allowed flags:

- Minus sign (`%-5.2f`) → left-justifies the content;
- Plus sign (`%+5.2f`) → prints a leading sign character, or right-justifies the content;
- Space (`%  5.2f`) → inserts a space before the value;
- Zero (`%05.2f`) → pads the printed output with zeroes;
- Pound (`%#5.2f`) → modifies numeric conversions.

5 – Minus vs. Plus Flags for Text

Including a Minus, as in `%-s`, left-justifies a text argument.

Including a Plus, as in `%+s`, right-justifies a text argument.

For example, consider the following print statement:

```
>> fprintf('%-8s\n%+8s', 'hello', 'hello')
hello
   hello
```

Command Window

5 – Plus Flags for Numeric Arguments

Including a Plus, as in `%+5.2f`, will print a leading sign.

For example, consider the following print statement:

```
>> fprintf('%+5.2f\n%+5.2f', 12.3, -12.3)
+12.30
-12.30
```

Command Window

So, the Plus operator prints a leading sign for numeric values, but it right-justifies text.

5 – Zero Flag

Including a Zero, as in `%05.2f`, pads the printout with zeroes.

For example, consider the following print statement:

```
>> fprintf('%06.1f', 12.3)
0012.3
```

Command Window

The Zero flag fills in the remaining space in the Field Width, which is discussed in several slides.

5 – Pound Flag

Including a Pound, as in `%#5.2f`, modifies numeric printouts.

This flag is more complicated and will not be included in the required content of this course.

You are welcome to read more about its use on Mathworks' website.

5 – String Formatting — Field Width

The Field Width specifies how many total characters the formatted output will occupy.

It cannot be lower than the minimum, but it *can* be increased. Take the following example:

```
>> fprintf('<%e><%6e><%14e><%f><%14f>', pi*50*ones(1,5))
<1.570796e+02><1.570796e+02><  1.570796e+02><157.079633><    157.079633>
```

Command Window

5 – String Formatting — Field Width

The Field Width specifies how many total characters the formatted output will occupy.

It cannot be lower than the minimum, but it *can* be increased. Take the following example:

```
>> fprintf('<%e><%6e><%14e><%f><%14f>', pi*50*ones(1,5))
<1.570796e+02><----------------------------------------------------------
```

Command Window

The **first** printout is the default for exponential notation.

5 – String Formatting — Field Width

The Field Width specifies how many total characters the formatted output will occupy.

It cannot be lower than the minimum, but it *can* be increased. Take the following example:

```
>> fprintf('<%e><%6e><%14e><%f><%14f>', pi*50*ones(1,5))
--------------<1.570796e+02>-----------------------------------------
```

Command Window

The **second** printout attempts to lower the width, and fails.

5 – String Formatting — Field Width

The Field Width specifies how many total characters the formatted output will occupy.

It cannot be lower than the minimum, but it *can* be increased. Take the following example:

```
>> fprintf('<%e><%6e><%14e><%f><%14f>', pi*50*ones(1,5))
----------------------------<  1.570796e+02>-------------------------
```

Command Window

The **third** printout increases the width and succeeds.

5 – String Formatting — Field Width

The Field Width specifies how many total characters the formatted output will occupy.

It cannot be lower than the minimum, but it *can* be increased. Take the following example:

```
>> fprintf('<%e><%6e><%14e><%f><%14f>', pi*50*ones(1,5))
-----------------------------------------<157.079633>----------------
```

Command Window

The **fourth** printout is the default for fixed point notation.

5 – String Formatting — Field Width

The Field Width specifies how many total characters the formatted output will occupy.

It cannot be lower than the minimum, but it *can* be increased. Take the following example:

```
>> fprintf('<%e><%6e><%14e><%f><%14f>', pi*50*ones(1,5))
-----------------------------------------------------------<    157.079633>
```

Command Window

The **fifth** printout increases the width and succeeds.

5 – What is Precision

Our work often has real-world implications, where precision is dictated by the physical limitations of the machinery we use.

Additionally, computers represent numeric values using the floating point format. There is inherent imprecision.

For **single** precision, they are precise to the order of **7** digits.
For **double** precision, they are precise to the order of **15** digits.

5 – String Formatting — Precision

These facts about computer precision are important when we interpret our work.

The Precision in string formatting specifies the number of decimals to print. Take the following example:

```
>> fprintf('<%.1f><%.5f>', 9.876, 9.876)
<9.9><9.87600>
```

Command Window

The **1**st rounds off at the tenths. The **2**nd prints 5 decimals.

5 – String Formatting — Subtype

And finally, the Subtype specifies that an input is a floating-point value when using the octal, hexadecimal, and decimal conversion characters.

Those conversion characters will not be included in the instruction material for this course.

You are welcome to read more about its use on Mathworks' website.

6 – Summary

This lecture covered:

- ✓ What it means to write a Hello World

  The Hello World script is the simplest demonstration of executing code in a new programming language.

- ✓ Using the Command Window for basic code execution

  Display commands can be executed directly in the Command Window to demonstrate printing Hello World text.

6 – Summary

✓ How to write a simple script in the Editor

The text display commands previously executed in the Command Window can also be written into a script. It is also possible to pass variables into a text display command.

✓ How to format variables in output messages

MATLAB includes formatting codes to inject variables into print output. This is demonstrated using an example of log messages executed throughout a script to keep track of important quantities.

6 – Summary

✓ The various formatting codes for print statements

Each of the formatting code areas was reviewed in detail. The purpose and functionality of the Identifier, the Flags, the Field Width, the Precision, and the Subtype were all covered to demonstrate how to effectively format output messages.